Student Name: **Student id:** **Sect#:** **Serial #:**

QUESTION #	1	2	3	4	TOTAL
MAX POINTS	10	12	17	14	
POINTS EARNED					

University of Bahrain College of Information Technology
Department of Computer Science

ITCS332: Organization of Programming Languages **SECOND TEST** **Date: MAY 4, 2015**
****

QUESTION ONE: **[5+5 pts]**

a) What will be printed after executing the following C++ code?

```
static int t=5;
void funX()
{ static int x1 = 4;
  int f = x1 * t;
  int *p = new int(5);
  x1 = x1 + t - *p;
  cout << x1 << '\t' << f << endl;
       t++; *p = *p + 2;
  delete p;
}
int main ()
{
    funX();
    cout << t << endl;
    funX();
```

```
4    20
6
5    24
```

b) Consider the following Ada-like code. The value of n printed by **print(n) :**

```
Procedure main is
   n: integer ;
   Procedure sub1 is
   begin
     n = n + 17 ; print (n) ;
   end ;
    Procedure sub2 is
      n : integer;
    begin
       n = 14 ; sub1 ; n = n + 5;
    end ;
begin
    n = 30 ; sub2 ;
end ;
```

1) Under static -scoped rules is **30+17= 47**

2) Under dynamic-scoped rules is **14+17= 31**

QUESTION TWO: **[8+4 pts]**

Part #1

1) Prolog operates in two modes *entry* and *query* mode.

2) The process of assigning temporary values to variables to allow unification to succeed is called *instantiation*. The process of finding useful values for variables in propositions that allows matching process to succeed is called *unification*.

3) The process that works on all subgoals of a given goal in parallel is called *breadth-first search*. The process of finding a complete proof for the first subgoal before working on others is called *depth-first search*.

4) The prolog query *?- [b, 34, 5|T] = [b, 34, 5, 66, b].* produces `T=[66,b]`

5) The prolog query *?-* `[[cat,X]|Y] = [[cat,black],[dog,rat],lion].` produces `X = black,` and `Y = [[dog, rat],lion].`

```
myst([],0).
myst([X|Y], R):- myst(Y,U), R is U+1.
```

6) The prolog query *?- myst([7, 4], 6,2,3, [8,-9],*`11,[-2]]`*, T).* produces *T= 7.*

Part #2

7) Write Prolog predicate(s) named "***most***" that displays the last item in a given list as shown in the following queries.

```
?- most([-5],T).
T = -5.

?- most([1,10,20],X).
X = 20.

?- most([1,[10,20],8,[7,5,-1]],U).
U = [7,5,-1].
```

most ([X], X).

most ([_ | T], R) :- most (T, R).

QUESTION THREE: **[17 pts]**

Part #1 **[6 pts]**

1) List 2 differences between stack-dynamic and heap-dynamic arrays. **[2 pts]**
   a) Stack-dynamic arrays are allocated on the **stack**, while the heap-dynamic arrays are allocated on the **heap**.
   b) Heap-dynamic arrays **can grow and shrink**, while stack-dynamic arrays **can not**.
2) Give one advantage and one disadvantage of long names. **[2 pts]**
   a) **The advantage is** that they are **meaningful and improve the readability**.
   b) **The disadvantage is** that they are **wasting the memory space of the symbol table.**
3) In C++, the main disadvantage of stack-dynamic variables is large execution time. Justify. **[1 pts]**
   Execution time is wasted for **allocating stack-dynamic variables** on every entry of the defining block and for **deallocating them on every exit** from the defining block.
4) What is the difference between a keyword and a reserved word? **[1 pts]**
   keyword can be redefined by the programmer and the **reserved word** cannot be redefined.

Part #2 **[11 pts]**

1) In C++, the base address = `1250` and "`double U[220];`"
   The address of the element `U[80]` = **`1250 + (80-0)*8.`**
2) In C++, the base address =`3200` and `short x[80][60];` the address of `x[17][24]` is **`3200 + 2 *[(17-0)*60 + (24-0)].`**
3) The dynamic length strings are implemented using **`Linked Lists`** or **`Adjacent memory Cells.`**
4) In programming languages, aliases are created using **`references`** or **`parameters`**.
5) A data type is a **`collection of data values (objects)`** plus **`a set of predefined operations that can be applied on those objects.`**
6) In C++, the static scope is created for every **`block`** and **`function`**.
7) The index type in any array reference is decided by **`Language Designer.`** The storage size of a given data type is specified by **`Language Implementer.`**
8) For multidimensional arrays, using row-major or column-major ordering of elements is decided by **`Language Implementer.`** The code to access an array element is generated by the **`Compiler.`**
9) The two major disadvantages of static variables are: **`They do not support recursion`** and **`Low efficient use of memory space.`**
10) The two disadvantages of dynamic length string implemented as a linked list are**`: complex and slow string operations`** and **`pointer space overhead`**.
11) Name two character string design issues: **`string type: primitive or structured`** and **`string length: static or dynamic`**.

QUESTION FOUR: Study the following C_like code and answer ALL questions below: **[14 pts]**

```
65)  static int z[64];
66)  void funA(double size)
67)  {   int w[64];      int y[size];
68)      int x[]={10,15,-24,-88,77};
69)      int *f = new int[24];
         ... ... ... ...
77)      sqrt(z[10]);

         ... ... ... ...
85)  }
86)  void char funB()
87)  {   char *uptr = new char('U');
88)      static float d = -4.75;
89)      double sum; ... ... ... ...
100) }
```

1) The scope of a array ***x*** is ***from line #68 in function funA to line #85.***
2) The scope of a variable ***d*** is ***from line #88 in function funB to line #100.***
3) The type of a pointer variable **uptr** is ***stack-dynamic***
4) The type of the object (variable) pointed to by **uptr** is ***explicit-heap dynamic***
5) The lifetime of a variable **d** begins when **the function *funB* is loaded first time** and ends when **the entire *program* terminates.**
6) The array ***f*** is allocated space ***on the heap*** of program memory, while the array ***w*** is allocated space ***on the stack.***
7) The array ***z*** is bound to storage during the ***loading*** time of the program. The call to the standard function ***sqrt*** in function ***funA*** is bound to its code at the ***linking*** time.
8) The type of array ***f*** is ***fixed-heap dynamic***
9) The type of array ***w*** is ***fixed stack-dynamic.***
10) The type of array ***y*** is ***stack-dynamic***
11) The type of array ***z*** is ***static***
12) The lifetime of array **z** begins when ***the program is loaded*** and ends when **the entire *program* terminates.**
13) The lifetime of a variable **sum** begins whenever ***the function funB is called*** and ends whenever ***the function funB is terminated (return).***
14) The lifetime of a variable **uptr** begins whenever ***the function funB is called*** and ends whenever ***the function funB is terminated (return).***